A Three-Layer Architecture for E-Contract Enforcement in an E-Service Environment

Dickson K.W. Chiu[1], S.C. Cheung[2], Sven Till[2]
[1]*Department of Computer Science and Engineering, Chinese University of Hong Kong, Shatin, HK*
[2]*Department of Computer Science, Hong Kong University of Science and Technology, Kowloon, HK*
email: kwchiu@acm.org, {scc,till}@cs.ust.hk

Abstract

In an e-service environment, contracts are important for attaining business process interoperability and enforcing their proper enactment. An e-contract is the computerized facilitation or automation of a contract in a cross-organizational business process. We find that e-contract enforcement can be divided into multiple layers and perspectives, which has not been adequately addressed in the literature. This problem is challenging as it involves monitoring the enactment of business processes in counter parties outside an organization's boundary. This paper presents an architecture for e-contract enforcement with three layers, viz., document layer, business layer, and implementation layer. In the document layer, contracts are composed of different types of clauses. In the business layer, e-contract enforcement activities are defined through the realization of contract clauses as business rules in event-condition-action (ECA) form. In the implementation layer, cross-organizational e-contract enforcement interfaces are implemented with contemporary Enterprise Java Bean and Web services. We present a methodology for the engineering of e-contracts enforcement from a high-level document-view down to the implementation layer based on this architecture, using a supply-chain example. As a result, e-contracts can be seamlessly defined and enforced. Conceptual models of various layers are given in the Unified Modeling Language (UML).

1. Introduction

The Internet has now become a global common platform where organizations and individuals communicate with each other to carry out various commercial activities and to provide value-added services. The term "e-service" refers to a service provided over the Internet. The wide adoption of e-services, however, poses a challenging problem to the enforcement of contracts across organizations. This is because an architecture that allows an organization to control or monitor the business processes of its counter-parties is not generally available.

A contract is a binding agreement between two or more parties, defining the set of obligations and rewards in a business process. An *e-contract* is a contract in electronic format, regulating cross-organizational business processes over the Internet. As e-services become more popular, widespread use of e-contracts in the business world is expected. The ability of an e-service system to readily create e-contracts with enforcement measures will soon become a critical success factor for the provision of e-services. This is particularly applicable to standard business interactions that could take place over the Internet, such as the purchase and sale of goods. New e-contracts for these business interactions can be defined based on standard *contract templates*, so that the effort in development and support of the contract's whole lifecycle (such as negotiation, enactment and enforcement) can be streamlined and reused.

Specific business interactions not covered by the clauses found in standard contract templates can be provided as *contract variations* or *contract escalations*. A contract template is the reference document that forms the basis on which a new contract is negotiated. A contract template consists of a number of *contract clauses*, each addressing a specific concern in the business interaction. Each contract clause contains a set of *template variables* whose values are to be negotiated in order to create a customized contract. The following example illustrates a contract clause in a sales contract template, where the brackets identify template variables in the clause.

> "*The PURCHASER shall send a Letter of Credit for the GOODS to the SUPPLIER in the currency of [ ] within [ ] days of the invoice date. The SUPPLIER shall on receipt of the Letter of Credit ship the GOODS to the PURCHASER within [ ] days and provide the PURCHASER with shipment details.*"

We have done some preliminary work [9] on the feasibility of modeling composite e-contracts based on

cross-organization workflows with workflow views. We have also studied the engineering of e-contracts for its enactment [4][5]. During these studies, we identified an acute need for a concrete methodology that allows an e-contract to be seamlessly analyzed from its textual documentation to its enforcement over the Internet. To address this, we propose to structure an e-contract in multiple levels and perspectives, viz., document layer, business layer and implementation layer. Conceptual models of these layers can be expressed uniformly in the Unified Modeling Language (UML), a widely accepted notation in object-oriented modeling [22]. We believe that the life cycle of an e-contract should be similar to that of a software system, i.e., definition, analysis and realization. This approach facilitates the understanding of an e-contract from its fundamentals to its implementation, which has been illustrated in earlier works [4].

A crucial task of this kind of implementation is the e-contract enforcement, in particular the monitoring, which has not been adequately addressed in the literature. We distinguish contract enforcements that address concerns about "what" is to be fulfilled in a contract from contract enactment that is concerned with "how" to fulfill a contract. The former deals with the detection and handling of contract breaches and exceptions while the latter deals with "normal" enactment of business processes. As such, contract enforcement can be considered as a conformance testing of contract enactment against a contract from a software engineering viewpoint. This problem is particularly challenging, among other electronic contracting activities, as it involves monitoring the enactment of business processes in counter parties *outside* an organization's boundary.

The contributions and coverage of this paper are: (i) a meta-model of e-contracts and e-contract templates, (ii) a three-layer architecture for cross-organizational e-contract enforcement, (iii) a methodology for elicitation of e-contract enforcement based on this multiple layer architecture, and, (iv) a feasible implementation outline for e-contract enforcement with Enterprise Java Bean (EJB) and Web services.

1.1. A Three-layer Architecture for E-Contract Enforcement

An e-contract defines clearly the requirements of business processes and the roles to be played by the parties involved. This definition is subject to analysis that aims to (a) identify the relations between the involved business entities, (b) the events or actions that take place in different parts of the business processes, and (c) the exceptions and possible contract breaches that may arise. Finally an e-contract is realized and enacted using existing Internet technologies, such as Web services [23] and EJB [27].

Depending on their job responsibilities, users across an organization may have different perspectives regarding an e-contract. For example, an implementation model of an e-contract that contains details of an implementation in Web services may not provide managers with information at the right level of abstraction. Instead, a business layer with information about rules and actions is more appropriate. It is more relevant to a system analyst who needs to refine an e-contract into a system design for subsequent enforcement. To allow for reusability and extensibility, a layered architecture of e-contracts is formulated in an object-oriented model using UML.

Layer	*Artifacts*
Document	Meta-model for e-contracts and templates: Contract clauses (Obligation, Permission, Prohibition) and Parties
Business	Meta-model for e-contract enforcement: Business events, Business rules, Business actions and Business entities
Implementation	Action implementation (Enterprise JavaBeans components) Cross-organizational interface (Web services XML schemas)

Table 1: An Architecture for E-Contract Enforcement

In the *document layer*, contracts are composed of different types of clauses, which typically include *obligation*, *permission* and *prohibition* [20]. A complex contract clause may consist of simpler clauses and relate to other contract clauses. A contract involves *parties*, together with their roles in the contract. The document layer corresponds to our meta-model for e-contracts and templates as detailed in Section 2.

The *business layer* of an e-contract specifies an e-contract from a business process point of view. It comprises four parts, viz., *business rules*, *business events*, *business actions* and *business entities*. *Business rules* specify the clauses of the contract in an ECA-rule paradigm. These rules are triggered by *business events*. *Business actions* capture the details of the activities required in the contract, including the set of roles involved in each activity and its consequences in terms of generated resultant events. *Business entities* are the set of data objects (including documents, etc.) relevant to the e-contract. The business layer will be revisited in more detail in Section 3.

The *implementation layer* of an e-contract enforcement comprises two parts, viz., action implementation, and cross-organizational event interface, and is based on contemporary Enterprise Java Bean (EJB) [27] and Web services [10] technologies. We choose the implementation of each action to be carried out by

computer systems for e-contract enforcement in EJB components because it supports three-tier implementation, is highly object orientated and component based, and is available for any platform. For the cross-organizational event interface, we employ Web services [10] interface definitions for the required communications and interactions, in which XML schemas [30] among business entities are designed for this purpose. The advantage of using Web services is to establish cross-organizational collaboration via existing Internet standards, supporting both human web-based interactions and automatic programmed interactions. The implementation layer is detailed in Section 5.

A summary of our three-layer architecture of e-contract enforcement is given in Table 1. The rest of our paper is organized as follows. Section 2 introduces our meta-model for electronic contracting. Section 3 presents the requirements for e-contract enforcement together with a system architecture and meta-model to facilitate this. Section 4 presents the transformation of e-contract clauses in the document layer to contract enforcement rules in the business layer. Section 5 discusses the implementation layer based on Web services and EJB, followed by a comparison with related work in Section 6. Finally, we conclude the paper with ongoing research work in Section 7.

2. A Meta-model for Electronic Contracting

Let us first introduce a meta-model for e-contracts in UML [22] and then discuss the lifecycle of an e-contract.

2.1. A Meta-model of E-Contract Templates in UML

UML is a modeling language for visualizing, specifying, constructing, and documenting the artifacts based on an object-oriented paradigm. The language offers a standard way to write a system's blueprints, including conceptual things such as business processes and system functions as well as concrete things such as programming language statements, database schemas, and reusable software components [22].

Figure 1 presents our meta-model of an e-contract template that forms the basis on which an e-contract is refined. A template consists of a number of *contract clauses*; each concerns some of the *parties* to be bound by the e-contract. Typical contract clauses can be divided into three types of *contractual constraints*: *obligation*, *permission* and *prohibition* [20]. For example, a customer is obliged to pay according to the payment terms and a supplier is not allowed to cancel the order once committed. A complex contract clause may consist of several simpler clauses or refer to other clauses. In an e-contract template, a contract clause may contain a number of *template variables*, such as the product, price and quantity. For each contract instance, these variables are to be refined in an e-contract through negotiations and finally agreed upon a set of *accepted values*.

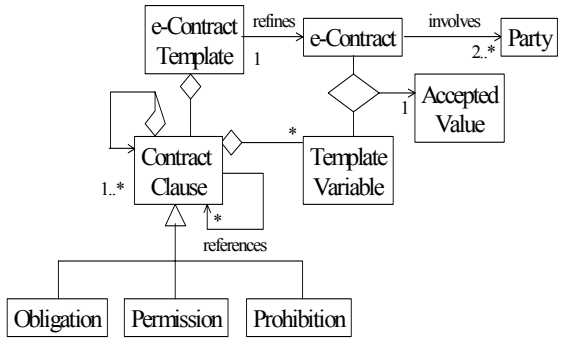

Figure 1: Meta-Model of an E-Contract Template in UML

Based on the previous contracts prepared and experience, a business can abstract common clauses and differentiate the parameters to create contract templates, according to this meta-model. This template provides a basis on which the whole contract lifecycle, as described in the next section, may take place. The most common contract being used in business is probably a sales contract. Figure 2 gives an example sales e-contract template instantiated from the meta-model in Figure 1. The sales e-contract consists of four contract clauses; each in turn contains one or two template variables. For example, unit price, quantity and delivery date are variables.

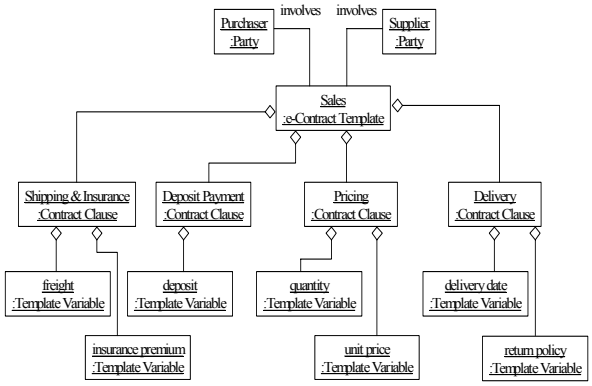

Figure 2: A Sales E-Contract Template as an Instance of the Meta-model in Figure 1

2.2. E-Contract Lifecycle

For each e-contract, the whole lifecycle of electronic contractual activities (as illustrated in Figure 3) involves not only contract enactment and contract enforcement, but also pre-contract activities, such as exchanging business information and contract negotiation. Exchanging business information includes advertisement activities of the service provider (push mode) and information

collection and comparison by potential customers (pull mode). This can be performed via electronic platforms such as electronic marketplaces, portals, and brokers.

Once a customer and a service provider identify each other, negotiation is carried out. Negotiation is a decision process in which two or more parties make individual decisions and interact with each other for mutual gain. We have proposed a contract template driven approach to this process [7], involving the negotiation of template variables, in order to avoid uncontrolled openness of issues, thus improving the effectiveness of negotiation.

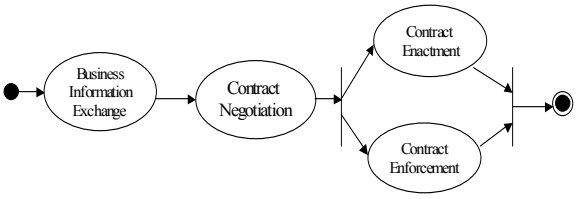

Figure 3: E-Contract Lifecycle

3. E-Contract Enforcement

In this section, we first discuss the requirements for e-contract enforcement. Then we introduce our enforcement architecture and give an overview of the transformation process towards an implementation for e-contract enforcement, focusing on the monitoring part.

3.1. Requirements for E-Contract Enforcement

The enforcement of a contract consists of two main issues, viz., the recognition and the handling of contract breaches. In order to recognize a contract violation, the compliance of a contract has to be kept under constant surveillance. Otherwise, contract violation cannot be recognized and the aggrieved party cannot react to the breach. How this breach should be handled depends on the aggrieved party and on the degree of the damage. The form of reaction ranges from ignoring the breach via invoking an exception rule through to a human intervention. Enforcement can be summarized by: *enforcement* is *monitoring* plus *handling*.

As mentioned above, an e-contract is composed of clauses. Each clause represents one of the three types of *contractual constraints*, viz., *obligation*, *prohibition* and *permission*. A contract defines the responsibilities and duties of the involved parties of a business process. Clauses state conditions where exceptions occur. Therefore they are used to cross check and to enforce the mutual agreement of the different business parties. In order to enforce an e-contract, many variables such as the status of delivery or the response time of an e-service have to be monitored. These variables may include confidential information, for example, balances of bank accounts or credit cards' numbers. One approach is to launch an enforcement service that constantly checks the validity of all these variables (according to the contract clauses). However, this incurs tremendous overheads to a system, and this mechanism is not practical to be extended across organizational boundaries. Alternatively, motivated by active database paradigms [11], the transformation of contract clauses into ECA rules can systematically reduce the monitoring effort. Now the monitor becomes only active when an interesting event occurs. Interesting events are to be raised by each party or some information sources. The demand of resources to enforce the contract is greatly reduced by using ECA rules because the amount of surveyed variables at one time is much less and the monitoring software is not permanently active.

It should be noted that information provided by sources, which are not directly involved in the contract, might be of interest to one of the contract partners, e.g., a news channel broadcasts a message about an earthquake in Taiwan that has damaged a semiconductor factory. This is particularly important to a computer manufacturer because the prices for chips, in particular memory chips and microprocessors, may rise soon. In that case, the manufacturer may decline to accept large orders based on an old price.

3.2. An Architecture for Cross-Organizational E-Contract Enforcement

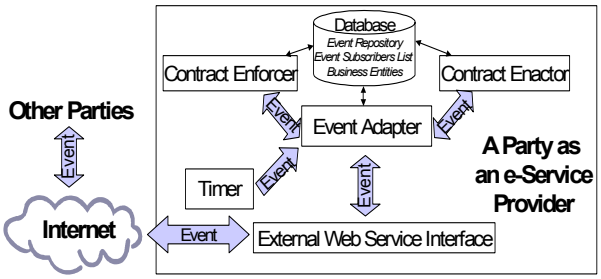

Figure 4: An Architecture for Cross-Organizational E-Contract Enforcement

Figure 4 depicts an architecture for cross-organizational e-contract enforcement based on the requirements discussed in the previous subsection. Each e-service provider hosts a *contract enactor* subsystem to perform regular business activities for service contract enactment. It also hosts a separate *contract enforcer* subsystem to detect contract breaches and to trigger relevant business actions upon such breaches (or other exceptions). Events are published and subscribed through an *event adapter*. The event adapter collects internal events from the contract enactor and external events from the *external Web service interface.* Events collected are filtered and transformed to a structure accepted by the event enforcer. Temporal events are generated by a *Timer* subsystem. In addition, each party maintains a database,

which stores the business entities, event repository (event log) and event subscriber lists. The advantage of this architecture is its support of a flexible peer-to-peer model. It does not require a central facilitator or moderator.

3.3. A Meta-model of Contract Enforcement

Figure 5 presents our meta-model for cross-organizational contract enforcement in UML. A business process can be modeled as a set of actions to be executed by a set of parties, each playing certain roles in the process. Typically, an action is recursively decomposed into *sub-actions* and eventually down to a unit level called *tasks*. In subsequent discussions, an action is a workflow or sub-workflow that is performed by a *single* party. For example, the action Check System Config is carried out by the system integrator. It consists of two tasks: (i) to receive a quotation request from an end user and (ii) to validate the system configuration required in the request. The parties in a cross-organizational workflow may belong to different organizations.

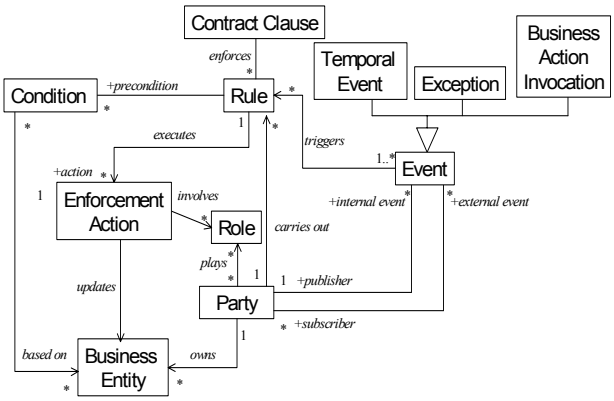

Figure 5: A Meta-model of Contract Enforcement in UML

An *event* occurs when something of interest happens to the system itself or to the user's applications. The source of events can be *internal* or *external*. An *internal event* originates within the organization that receives the event, while an *external event* originates from another organization. Examples of external events include the receipt of a request for quotation or of a purchase order. When an event occurs, it triggers some rules and the *condition* parts of these rules will be evaluated. Conditions are logical expressions defined on the states of business entities, such as the status of an order. Only if the *condition* is satisfied, the *enforcement action* part, which is a workflow, will be executed and may lead to other events, such as exceptions. The semantics of ECA rules for contract enforcement can be summarized as the following: *On* event *if* condition *then* enforcement action.

In a cross-organizational context, when an organization detects an *internal event*, it must explicitly send the event to other target organizations in order to notify them. This event is then considered to be an *external event* of each target organization. However, the target organizations must have subscribed for it beforehand. Examples of such events include change in delivery date or change in price. It should be noted that the subscription can be implicit. For example, placement of an order implies a subscription to the event of change in delivery date, as this is an obligation of the supplier.

Data required in the business process are encapsulated by *business entities* owned by some *party* participating in the workflows. The *enforcement actions* in the rules may update the state of these business entities, which in turn may trigger other *events*. Note that *exceptions* are special *events*, which deviate from normal expected behavior or prevent normal process execution. A business process often defines extra rules describing exceptions and their handling, and in our context, for contract enforcement. Actions may also be triggered by time-related elements, such as deadlines and durations. These elements can be represented by *temporal events* in the meta-model. In addition, *business action invocations*, such as those, that could possibly breach a contract (cf. section 4.2 on enforcing prohibitions), are also modeled as events in the meta-model.

4. E-Contract Enforcement Business Rules - from Contract Clauses to ECA Rules

To facilitate enforcement, contract clauses expressed in a format based on the meta-model in Figure 1 are analyzed and then transformed into a set of ECA rules, based on the meta-model in Figure 5. This set of ECA rules collectively formulates an operational model of the contract clauses for subsequent enforcement. We conducted a study based on a service agreement referred to as the "Terms and Conditions of Sale, Service and Technical Support" at the official website of Dell (Hong Kong) [12]. In this agreement, Dell plays the role of a system integrator and the customer the role of an end user.

Enforcement rule / Clause type	Event	Condition	Action
Obligation	*onDay(deadline(BAO))*	*NOT occurred(BAO)*	*raise(exception(BAO))*
Prohibition	*onOccurred(BAO)*	*prohibitionCondition (BAO)*	
Permission		*NOT permitted(BAO)*	

Table 2: Basic Mapping of Contract Clauses into ECA rules

Table 2 summarizes our methodology to map different types of contract clauses into enforcement ECA rules,

which will be detailed in the following subsections. *BAO* stands for an object that encapsulates a business action whose execution triggers the object creation. Our methodology helps discover some typical problems that arise from the ambiguity of natural language and the autonomous nature of individual organizations. We also suggest some measures to overcome them during the discussion.

After the involved parties have agreed to a contract, analysis is conducted. The analysis is driven by a methodology mapping the three contractual constraint types of contract clauses, i.e., *obligations* (what a party must do), *prohibitions* (what a party must not do) and *permissions* (what a party can but is not obliged to do), into ECA rules. Common contractual wordings may provide additional hints in the analysis to identify the constraint type of a contract clause. For example, the term "shall" tends to imply an obligation, "may" a permission and "shall not" a non-obligation, i.e., a prohibition or a permission. Since natural language formulations (particularly in contracts) can be multifarious, further analysis in the clause structures is often necessary.

An alternative is to map these rules into a set of logical expressions in deontic logic, a class of formal logic [20]. A rule of deontic logic has the following formal structure:

> *Rule #: <role> [is] (obligated | forbidden| permitted) [to] [do] (<action> [before <condition>] | satisfy <condition>) [, if <condition>][, where <condition>] [, otherwise see Rule <#>]*

Unlike ECA rules, deontic logic was not designed to be executable and therefore not associated with well-defined operational semantics. For instance, the triggering event for an action is often omitted, making it difficult to determine the execution of logical expressions. In addition, the deadline of an action or a task is often not stated. However, this is important for the enforcement of obligations; otherwise a party may defer the obliged action indefinitely. An obligation without stating a deadline or an event before which the obliged action must have taken place may even imply enforcement is inapplicable. Sometimes, the deadline is implied due to standard practices of the business, governmental regulation, etc., and must be added explicitly by the analyst. All these kinds of ambiguities, once found, should be clarified and confirmed by both parties to avoid confusion or later unnecessary disputes, and should not simply be left in a rule.

4.1. Enforcing Obligations

Consider an ECA rule R_{obl} that formulates an obligation where a business action A_{obl} must be performed by a deadline T_{obl}. The obligation can be enforced using the following mechanism. Upon reaching the deadline T_{obl}, a temporal event is generated by the Timer. This triggers the contract enforcer to fire rule R_{obl} and execute the enforcement action to check if the obliged party has performed the required business action A_{obl}. This check can be achieved by, as for example, searching the log file for invoked actions or occurrence of related events. In the case of payment obligations, suitable events could be the acceptance of a payment receipt or a change in a bank account's balance. If the obligation has not been fulfilled, the *contract enforcer* raises an exception. Based on this mechanism, ECA rules for obligation enforcement can be formatted using the following predicates. Here, *BAO* is an object encapsulating the required business action with a deadline, denoted as *deadline(BAO)*. A temporal event is generated on the date of deadline, denoted as onDay(deadline(BAO)). The predicate *occurred(BAO)* holds if the business action has occurred. An exception, denoted as *exception(BAO)*, is raised as a result of the rule execution.

E: onDay(deadline(BAO))	An ECA Rule
C: NOT occurred(BAO)	for Obligation
A: raise(exception(BAO))	Enforcement

For example, for the contract clause: "*7.1 Dell shall deliver the Products to the place of delivery designated by Customer and agreed to by Dell as evidenced in Customer's invoice ("Place of Delivery")",* the corresponding enforcement ECA rule can be:

> *E: onDay(deadline(DELIVER))*
> *C: NOT occurred(DELIVER)*
> *A: raise(exception(DELIVER))*

The customer could monitor this obligation by checking the list of products delivered before and on the delivery date, denoted as *deadline(DELIVER)* in the rule. However, problems may arise if Dell has already sent the products but due to certain circumstances, they have not reached the customer yet. In this case, Dell could prove the product delivery, as for example, by providing the tracking number of the sent package. In fact, this should be done as soon as the package is sent, in order to improve customer relationships.

As mentioned above, there are two sets of ECA rules necessary to implement an e-contract - for the *enforcement* and for the *enactment*. Enactment rules are triggered to invoke necessary actions on time, while enforcement rules are triggered once deadlines of obligations has been reached. Since an obliged action may need some time to complete, the action must be triggered early enough, as for example, six days before the deadline. The following enactment ECA rule for the same contract clause illustrates this difference:

E: onDay(before(deadline(DELIVER), 6))
C: valid(place(DELIVER)) & ready(DELIVER)
A: perform(DELIVER)

We conclude this subsection by discussing a general problem of the impreciseness of natural languages. Phrases like "as soon as practicable" or "as soon as possible" are imprecise, lacking a concrete deadline. The handling of such ambiguity generally requires human attention. The analyst has to substitute these with concrete deadline in the formulation of ECA rules. For instance, consider the contract clause *"10.7 ...Dell shall respond to a request for such Emergency Service as soon as practicable after its receipt of such request. ..."* The corresponding enforcement ECA rule can be formulated in the following. Here, *N* is a chosen time allowance.

E: onDay(after(receiptDate(
EMERGENCY_REQUEST), N)))
C: NOT responded(EMERGENCY_REQUEST))
A: raise(exception(EMERGENCY_REQUEST))

4.2. Enforcing Prohibitions

The occurrence of a prohibited action (or prohibition) should be treated as an exception by the contract enforcer. Our meta-model in Figure 5 supports this scenario without any extension. One problem of the observation of prohibitions is that if a party performs a prohibited action, the party will probably try to hide or distract this fact as long as possible (unless the party does this by mistake or misunderstandings). Thus, in general, it will be quite difficult to observe or to recognize a prohibited action. Should it be easy to detect such an event of a prohibited action, the party probably would not invoke this specific action. This is a problem due to the autonomous nature of different organizations rather than that of our architecture or our model. A general ECA rule for prohibition enforcement can be described as follows:

E: onOccurred(BAO)
C: prohibitionCondition(BAO)
A: raise(exception(BAO))

Consider the contract clause *"14. Each party shall treat as confidential all information obtained from the other pursuant to a Contract which is marked 'confidential' or the equivalent or has the necessary quality of confidence about it and shall not divulge such information to any persons without the other party's prior written consent provided that this clause shall not extend to information which was rightfully in the possession of such party prior to the commencement of the negotiations leading to the Contract, ..."* For example, a corresponding enforcement ECA rule can be:

E: onOccurred(INFO)
C: confidential(INFO)
A: raise(exception(INFO))

However, if a party really passes confidential contract information to a third party, this is almost impossible to detect. Thus, the event "*onOccurred(INFO)*" is *non-monitorable*. On the other hand, Dell has a similar problem. Another contract clause states that the customer warrants buying the products only for its own internal use and not for re-sale. But Dell cannot easily check if the end-user buys the product for itself or to resell it to a third-party.

4.3. Enforcing Permissions

A permission is a temporary allowance to perform an otherwise prohibited action, i.e., a specific action may be carried out only within a certain allowed time period. Some actions may be permitted under specific *situations* (i.e., events plus conditions). Note that a party is not obliged to carry out a permitted business action. After the message of the invocation of a permitted action is received, the contract enforcer checks if the conditions for the permission are met or not. If the permission situations are not met, the contract enforcer raises an exception. Whether the actual action invocation will be interrupted or not depends on the exception handler. A general ECA rule for obligation enforcement can be formulated as follows:

E: onOccurred(BOA)
C: NOT permitted(BOA)
A: raise(exception(BOA))

For example, consider the contract clause *"2.1 ... Dell shall be entitled to refuse to accept orders placed by the Customer if the Customer breaches or Dell, on reasonable grounds, suspects that the Customer will breach this warranty."* A corresponding ECA-rule can be stated as follows where *REFUSE_ORDER* is the action object encapsulating the business action "refuse order".

E: onOccurred(REFUSE_ORDER)
C: NOT badlisted(customer(REFUSE_ORDER))
A: raise(exception(REFUSE_ORDER))

The event *onOccurred(REFUSE_ORDER)* can be observed by the customer upon the receipt of a order cancellation message. But the customer may have problems in understanding the applicability of the condition, as the internal criteria of Dell for trustable customer is not disclosed to the public.

Consider another contract clause "*3.1 Dell may, at its sole discretion, allow a Customer to cancel its order after acceptance at no charge, if written notice of such cancellation is received by Dell before commencement of manufacture of the Products. If Dell allows a Customer to cancel its order after manufacture but before shipment of the Product, Dell shall be entitled to levy a cancellation*

charge equal to 20% of the price of the Products." The clause may be formulated by the following ECA rule. The business action "levy cancellation charge" is encapsulated by an action object *LEVY*.

E: onOccurred(LEVY)
C: NOT (dateOfCancellation(order(LEVY)) > dateOfManufacture(order(LEVY)) & cancellationApproved(order(LEVY)))
A: raise(exception(LEVY))

A customer can hardly tell whether the commencement of manufacture of the product has already started when canceling the order. It is almost non-monitorable because normally a customer does not have access to such kind of internal data. However, Dell may improve the situation by informing the customer when the commencement starts through its enactment system. As such, the monitorability / enforceability of this specific permission would change from non-monitorable to monitorable.

4.4. Discussion

In this section, we have presented a methodology to map different types of contract clauses into enforcement ECA rules. We have also highlighted typical problems that can be discovered by our methodology, together with some measures of overcoming them. However, it is not possible to suggest general measures to *handle* contract breaches or exception, as these often involves domain specific knowledge, which are either explicitly specified in other contract clauses or implicitly regulated by laws and standards.

Some of the problems in analyzing contracts arise from ambiguity and impreciseness of natural languages. These might be sought out with reference to other laws, regulations, standard trade practices, etc. However, to avoid unnecessary disputes, the parties involved should discuss and clarify the matter, and if necessary, amend existing or forthcoming contracts accordingly.

Other problems arise from the autonomous nature of individual organizations. Events that need to be monitored often come from counter parties in other organizations, and might not be monitorable. Thus, cooperation and trust should be developed among trade partners to alleviate this problem. In general, this improves the transparency of operations, services, and in turn, customer relationships [8], and is therefore vital in contemporary e-service providers under strong competitions. On the other hand, such events may be made monitorable by adding explicit clauses in the contract to demand the provision of such events among the parties where appropriate. Alternatively, trade standards or e-services standards should be established accordingly to minimize such efforts and to streamline fair and effective monitoring of e-service provision in the digital economy.

5. An Implementation Outline for E-Contract Enforcement

Web services can be used to interface different enforcement and enactment systems within and across organizations by supporting appropriate cross-organizational communication and interoperability. In this section, we outline the implementation of cross-organization interfaces for e-contract enforcement through Web services. Details of e-contract enactment using Web services can be found in our previous work [4]. Based on the functional and data requirements of the *event adaptor*, three Web services, viz., for publishing events, for receiving events, for subscribing events, are identified as shown in Figure 6. Examples of these Web services are summarized in Table 3.

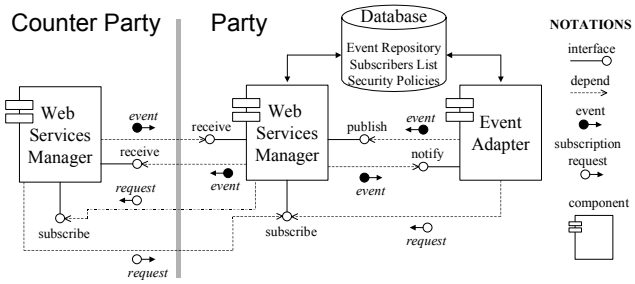

Figure 6: Web Services Implementation of an E-Contract Enforcement System

The *publish* Web service will be invoked by the *event adaptor*. The input parameter is the occurred event or exception. Based on this, the Web service checks the subscribers list and the security policies, and then notifies the valid subscribers. Notification can be performed via different kind of protocols like e-mail, fax, ICQ message, or even via another Web service. How the subscribers should be notified is specified in the subscription process via the *subscribe* Web service. The *subscribe* Web service registers requests for an event subscription including several parameters such as the requester, the subscribed event, and how the requester wants to receive the event notification. The *receive* Web service is used to receive subscribed events published by the counter party organizations. Received events are recorded at the Event Repository and forwarded to the Event Adapter, which in turn transforms them into the forms as required by the *Contract Enforcer* and *the Contract Enacter* in Figure 4.

In addition, a system integrator can offer more specific Web services like *takeOrder* or *trackOrder* as shown in Table 4. In order to provide a better service and

to increase the trust between the involved parties, the *trackOrder* Web service can be composed with another *trackDelivery* service, so that customers may acquire all necessary information from one provider.

Publish *Web Service*	**Subscribe *Web Service***	**Receive *Web Service***
Input: EventReceiving Acknowledge • EventReceiving Acknowledge Output: EventMessage • Date • Sender • Receiver • Event o Event name o Event type o Event subject o Event message body o Prio	Input: SubscriptionRequest • Eventprovider o Name o Address o E-Mail • SubscribedEvent • NotificationParameter o transmissionPort TransmissionParameter (like email | fax | icq no. |...) Output: SubscriptionResponse • SubscriptionResult	Input: EventNotification • Date • Sender • Receiver • Event o Event name o Event type o Event subject o Event message body • Prio Output: EventReceivingResponse • EventReceivingAcknowledge

Table 3: Sample Web Service Specifications for Contract Enforcement

takeOrder *Web Service*	**trackOrder *Web Service***
Input: OrderRequest • Buyer o Name o Address o E-Mail • ProductList o Product • Product ID • Product Name • Quantity • Price Output: OrderResponse • OrderResult o OrderNr o Password o Estimated Delivery Date	Input: OrderStatusRequest • OrderNr • Password Output: OrderStatus • Progress • Estimated Delivery Date • Optional: DeliveryNr

Table 4: Other Possible Web Services for Contract Enforcement

6. Related Work

Modeling of e-contracts can be dated back to the Contract Net Protocol [25]. However, they only concentrated on low-level transaction aspects. Gisler et al. [14] presented an architecture for legal e-contracts, but not a mechanism for modeling e-contracts. Grosof [18] introduced a declarative approach to business rules in e-commerce contracts by combining Courteous Logic Program and XML. Tan and Thoen proposed a conceptual view to represent the contents of business contracts with Formal Language for Business Communication (FLBC) for contract negotiation based on the event semantics. Marjanovic and Milosevic [20] modeled a contract with deontic logic, based on obligation, permission and prohibition. They proposed a contract monitoring mechanism through a trusted third party instead of a peer-to-peer model. Recently, Karlaplem et al. [19] proposed a meta-model of e-contracts with entity-relationship diagrams for generating workflows to support e-contract enactment, but they did not address enforcement issues.

Though there are many web-enabled WFMS research prototypes ([2], [21]) and commercial products ([3], [28], [25], [13], [18]), few of them address problems in e-contracts or cross-organizational workflow comprehensively. We have done some work on the E-ADOME [9] system, which proposed a novel concept of workflow view for cross-organizational workflow interoperability and e-contract development. We are also deriving a methodology for extending workflows beyond organizations by analyzing information (data plus events) requirements in a Web service environment [5].

CrossFlow [16] models virtual enterprises based on a service provider-consumer paradigm, in which organizations (service consumers) can delegate tasks in their workflows to other organizations (service providers). However, it does not observe contract provisions in legal context aspects. Furthermore, because it does not support ECA rule based on cross-organizational events and just can monitor workflow status (such as the start or the end of a task), frauds cannot be observed easily. The COSMOS project (Common Open Service Market for SMEs) [17] have developed Internet-based electronic contracting services to facilitate business transaction processes based on CORBA (Common Object Request Broker Architecture [24]). E-contracts are modeled as a combination of objects, which can be exchanged among different parties and stored in XML. Though workflow based contract enactment can be facilitated with another explicit flow-model, contract enforcement issues are not addressed..

In summary, previous work addressed either only specific portions of the e-contract enactment process without focus on enforcement aspects, or just some of the supporting facilities required for e-contract enforcement.

7. Conclusions

This paper has presented a meta-model for e-contracts and templates. We have also detailed a pragmatic architecture for cross-organizational e-contract enforcement comprising three layers, viz., document layer, business layer, and implementation layer. To demonstrate the feasibility of our architecture, we have presented an architecture and a methodology for developing e-contract enforcement rules, in an e-service environment, using a supplier's example. In particular, we have detailed how to analyze a contract at the *document layer* to define e-contract enforcement rules at the *business layer*. We have also highlighted typical problems that can be discovered by using our methodology, together with some measures of overcoming them. We finish our discussion with an outline of the *implementation layer*, which is facilitated by contemporary standard software technologies of EJB and Web services. As such, the development of a system for e-contract enforcement across organization boundaries

can be streamlined in the context of e-service providers and consumers.

At the same time, we are working on further details of process adaptation for interoperability, e-contract negotiation [6], methodologies for preventive measures avoiding contract breaches, and the use of workflow management systems for the whole e-contract lifecycle [7]. On the other hand, we are interested in the application of e-contracts and customer relationships management [8] in various advanced real-life e-service environments, such as supply-chain, procurement, finance, stock trading and insurance. We are developing a more unified way to exchange information, including workflow views, with other agents, through Web services.

Acknowledgment

The work is partially supported by the Hong Kong Research Grant Council with the research grant (ref. HKUST6187/02E).

References

[1] S. Abiteboul, A. Bonner. Objects and Views. *In Proceedings of ACM SIGMOD Conference*, pp. 238-247, 1991.

[2] T. Cai, P.A. Gloor, S. Nog, "DartFlow: A Workflow Management System on the Web using Transportable Agents", Technical Report PCS-TR96-283, Dartmouth College, Hanover, N.H., 1996.

[3] F. Casati, et al. Adaptive and Dynamic Service Composition in eFlow. HP Lab's Technical Report HPL-2000-39, March 2000.

[4] S.C. Cheung, D.K.W. Chiu, S. Till. A Three-Layer Architecture for Cross-Organization E-contract enactment. In *Proceedings of* Web Services, E-business and Semantic Web Workshop, (In conjunction with CAiSE 2002), May 2002.

[5] S.C. Cheung, D.K.W. Chiu, S. Till. A Data-driven Methodology to Extending Workflows across Organizations over the Internet. 36th Hawaii International Conference on System Sciences (HICSS36), Jan 2003.

[6] S.C. Cheung, P.C.K. Hung, D.K.W. Chiu, On the e-Negotiation of Unmatched Logrolling Views, in the Proceedings of the 36th Hawaii International Conference on System Sciences (HICSS-36), Jan 2003.

[7] D.K.W. Chiu, S.C. Cheung, P.C.K. Hung, A Meta-model for Contract Template Driven e-Negotiation Processes, in the Proceedings of 6th Pacific Asia Conference on Information Systems (PACIS'02), Tokyo, Japan, September 2002, pp. 854-868.

[8] D.K.W. Chiu, W. C. W. Chan, G. K. W. Lam, S. C. Cheung and F. T. Luk. An Event Driven Approach to Customer Relationship Management in an e-Brokerage Environment. *36th Hawaii International Conference on System Sciences (HICSS36)*, Jan 2003.

[9] D.K.W. Chiu, K. Karlapalem, Q. Li, E. Kafeza. Workflow Views Based E-Contracts in a Cross-Organization E-Service Environment. *Distributed and Parallel Databases*, Kluwer Academic Publishers, 12(2-3):193-216, 2002.

[10] V. Chopra, Z. Zaev, G. Damschen, F. Norton, C. Dix, P. Cauldwell, R. Chawla, K. Saunders, G. Olander, T. Hong, U. Ogbuji, M. Richman. Professional XML Web Services, Wrox Press, 2001.

[11] U. Dayal. Active Database Management Systems. *Proc 3rd International Conference on Data and Knowledge Bases*, pp 150-169, 1989.

[12] http://www.ap.dell.com/ap/hk/en/gen/local/legal_terms.htm

[13] Enix Consulting Limited. An Independent Evaluation of i-Flow Version 3.5, 2000 (available at http://www.i-flow.com).

[14] M. Gisler, K. Stanoevska-Slabeva, M. Greunz, Legal Aspects of Electronic Contracts*, In CAiSE*00 Workshop of Infrastructures for Dynamic Business-to-Business Service Outsourcing (IDSO'00)* Stockholm, 5 - 6 June 2000; pp.53-62.

[15] A. Goodchild, C. Herring and Z. Milosevic. Business Contracts for B2B. In *Proceedings of the CAiSE*00 Workshop of Infrastructures for Dynamic Business-to-Business Service Outsourcing (IDSO'00)* Stockholm, 5 - 6 June 2000; pp. 63-74.

[16] P. Grefen and Y. Hoffner. Crossflow – Cross-Organizational Workflow Support for Virtual Organization. In Proc of the Ninth International Workshop on Research Issues on Data Engineering: Information Technology for Virtual Enterprises (RIDE'98), 1998; pp. 90-91.

[17] F. Griffel. Electronic Contracting with COSMOS – How to Establish, Negotiate and Execute Electronic Contracts on the Internet. *2nd Int. Enterprise Distributed Object Computing Workshop (EDOC '98)*, pp. 46-55, 1998.

[18] B. N. Grosof, A declarative approach to business rules in Contracts: Courteous Logic Programs in XML*, Proceedings of the 1st ACM Conference on Electronic Commerce (EC99),* Denver, Colorado, USA, Nov. 3-5, 1999; pp. 68-77.

[19] K. Karlaplem, A. R. Dani and P. R. Krishna. A Frame Work for Modeling Electronic Contracts. *International Conference on Conceptual Modeling (ER2001).* pp. 193-207, Nov 2001.

[20] O. Marjanovic, and Z. Milosevic. Towards formal modeling of e-Contracts, Proceedings of 5th IEEE International Enterprise Distributed Object Computing Conference, pp. 59 –68, 2001.

[21] John A. Miller, et al. Recovery Issues in Web-Based Workflow. *Proceedings of the 12th International Conference on Computer Applications in Industry and Engineering (CAINE-99),* pp. 101-105, Atlanta, Georgia Nov. 1999.

[22] Object Management Group. Foreword UML specification 1.4, September 2001.

[23] N. Sankaran. Building Web service applications. *Windows Developers Journal*, 12(10): 8, 12-13, 16-18. Miller Freeman, USA, Oct. 2001.

[24] J. Siegel. CORBA 3 Fundamentals and Programming, 2nd Edition, Wiley, 2000.

[25] R. G. Smith. The contract net protocol: High Level Communication and Control in a Distributed Problem Solver, *IEEE Transactions on Computers* 29(12), December 1980, 1104-1113.

[26] Y.H. Tan, W. Thoen. Using Event Semantics for Modeling Contracts. Proceedings of the 35th Hawaii International Conference on System Sciences, pp. 2198-2206, 2002.

[27] http://java.sun.com/products/ejb/index.html

[28] TIBCO Software Inc., which has acquired InConcert Inc., http://www.tibco.com

[29] http://vsys-www.informatik.uni-hamburg.de/projects/cosmos/

[30] http://www.w3.org/XML/